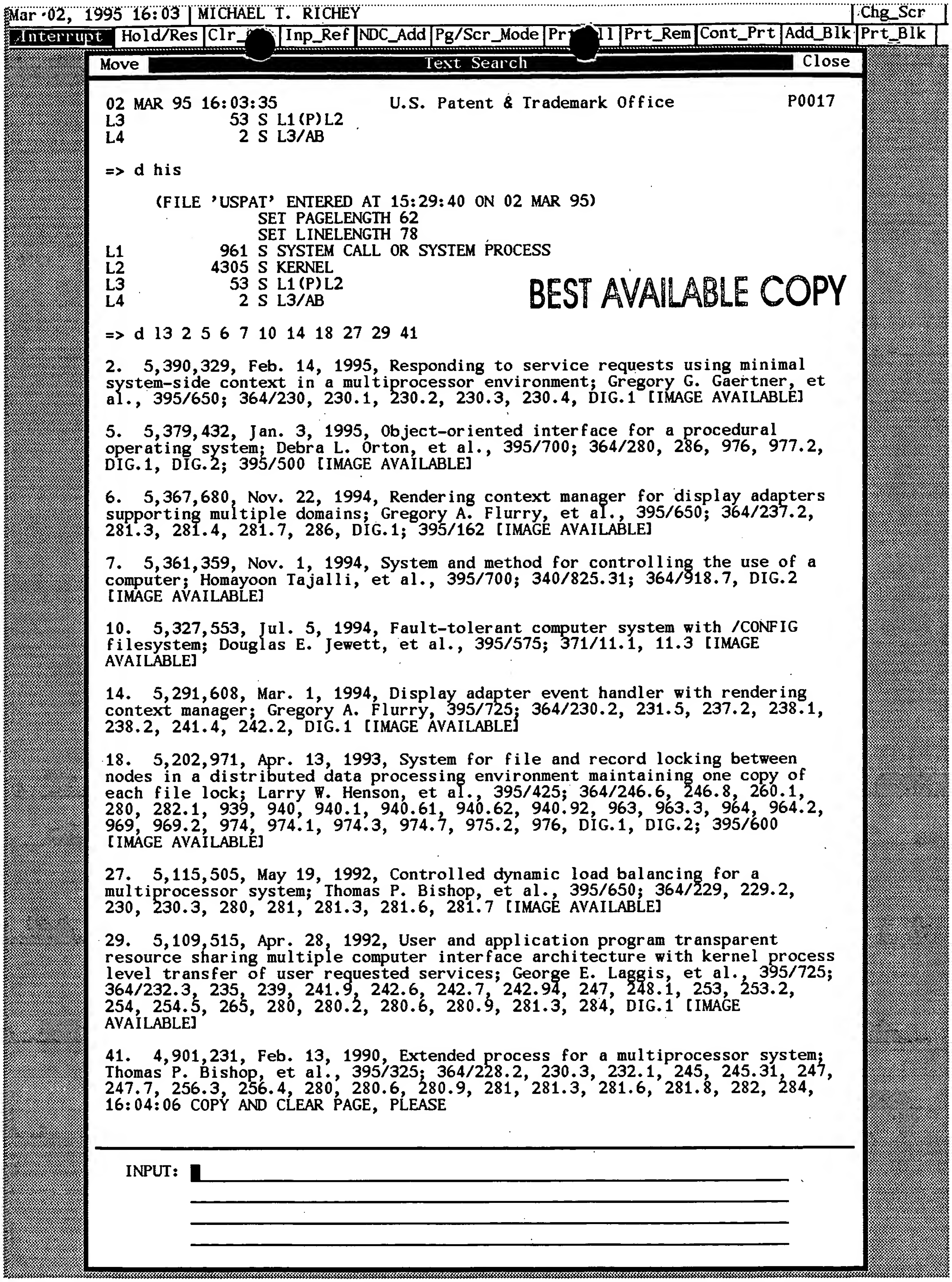
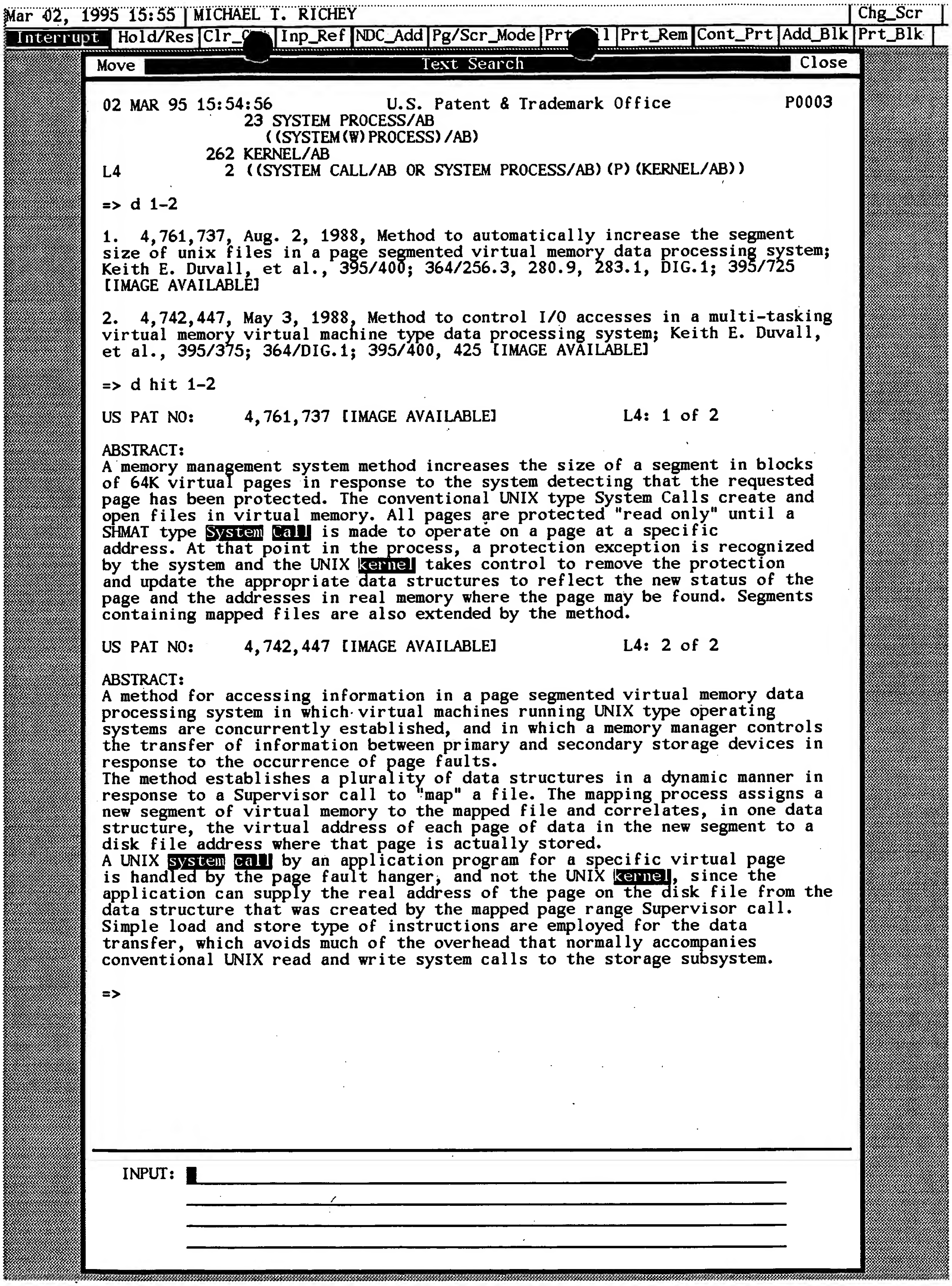
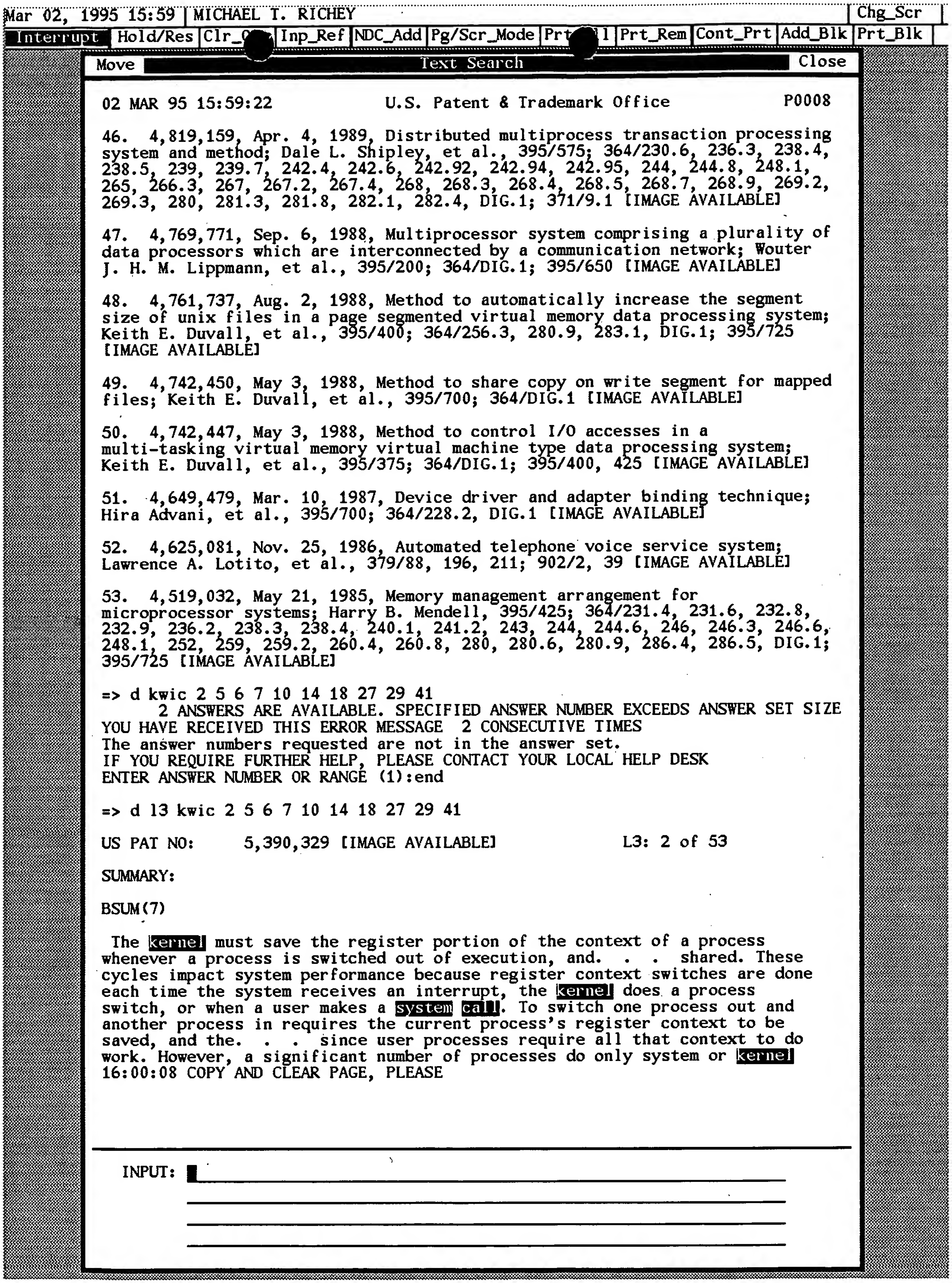


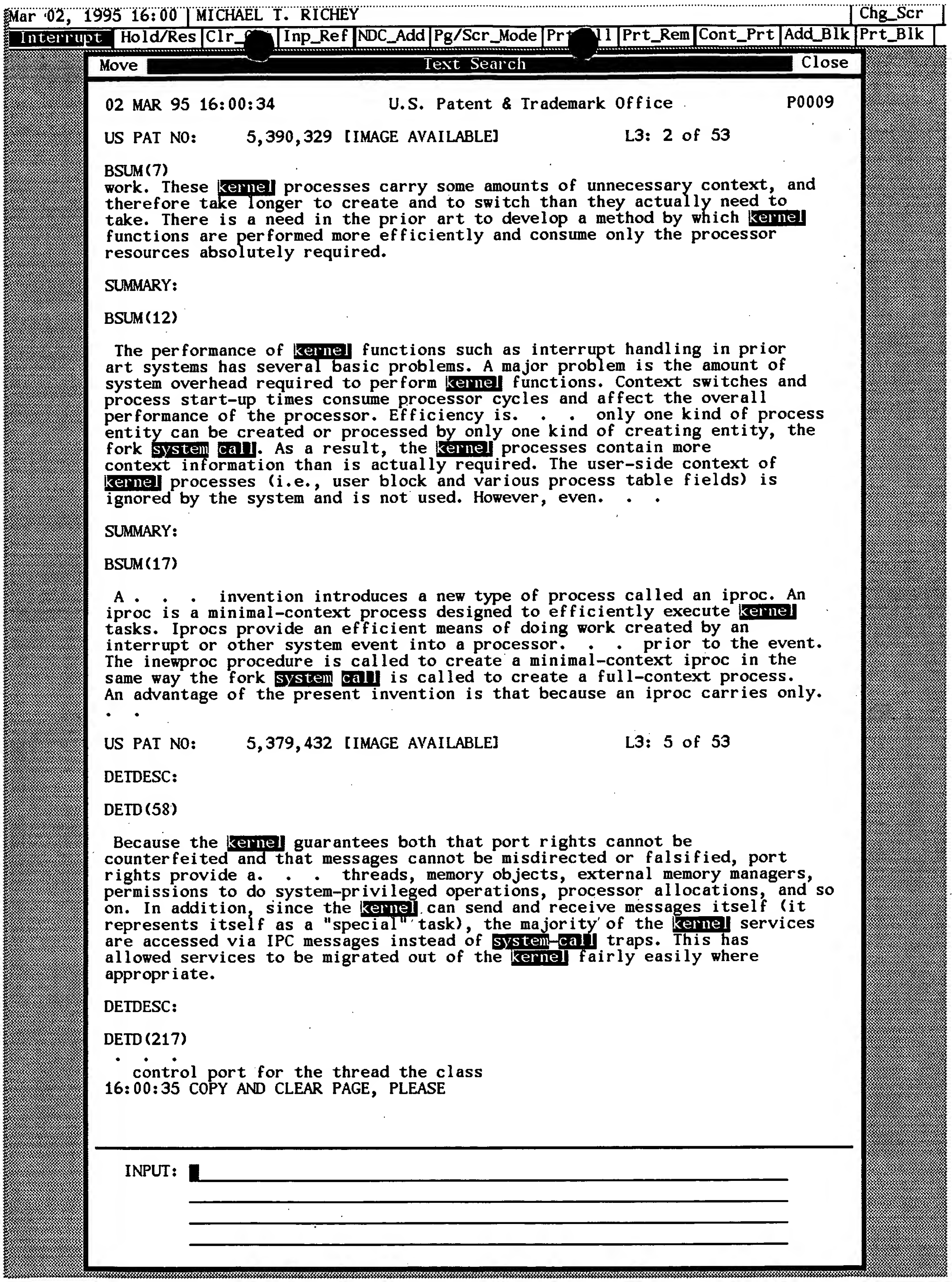
```
02 MAR 95 16:03:35          U.S. Patent & Trademark Office          P0017
L3            53 S L1(P)L2
L4             2 S L3/AB

=> d his

     (FILE 'USPAT' ENTERED AT 15:29:40 ON 02 MAR 95)
                SET PAGELENGTH 62
                SET LINELENGTH 78
L1           961 S SYSTEM CALL OR SYSTEM PROCESS
L2          4305 S KERNEL
L3            53 S L1(P)L2
L4             2 S L3/AB

=> d 13 2 5 6 7 10 14 18 27 29 41

2.  5,390,329, Feb. 14, 1995, Responding to service requests using minimal
system-side context in a multiprocessor environment; Gregory G. Gaertner, et
al., 395/650; 364/230, 230.1, 230.2, 230.3, 230.4, DIG.1 [IMAGE AVAILABLE]

5.  5,379,432, Jan. 3, 1995, Object-oriented interface for a procedural
operating system; Debra L. Orton, et al., 395/700; 364/280, 286, 976, 977.2,
DIG.1, DIG.2; 395/500 [IMAGE AVAILABLE]

6.  5,367,680, Nov. 22, 1994, Rendering context manager for display adapters
supporting multiple domains; Gregory A. Flurry, et al., 395/650; 364/237.2,
281.3, 281.4, 281.7, 286, DIG.1; 395/162 [IMAGE AVAILABLE]

7.  5,361,359, Nov. 1, 1994, System and method for controlling the use of a
computer; Homayoon Tajalli, et al., 395/700; 340/825.31; 364/918.7, DIG.2
[IMAGE AVAILABLE]

10.  5,327,553, Jul. 5, 1994, Fault-tolerant computer system with /CONFIG
filesystem; Douglas E. Jewett, et al., 395/575; 371/11.1, 11.3 [IMAGE
AVAILABLE]

14.  5,291,608, Mar. 1, 1994, Display adapter event handler with rendering
context manager; Gregory A. Flurry, 395/725; 364/230.2, 231.5, 237.2, 238.1,
238.2, 241.4, 242.2, DIG.1 [IMAGE AVAILABLE]

18.  5,202,971, Apr. 13, 1993, System for file and record locking between
nodes in a distributed data processing environment maintaining one copy of
each file lock; Larry W. Henson, et al., 395/425; 364/246.6, 246.8, 260.1,
280, 282.1, 939, 940, 940.1, 940.61, 940.62, 940.92, 963, 963.3, 964, 964.2,
969, 969.2, 974, 974.1, 974.3, 974.7, 975.2, 976, DIG.1, DIG.2; 395/600
[IMAGE AVAILABLE]

27.  5,115,505, May 19, 1992, Controlled dynamic load balancing for a
multiprocessor system; Thomas P. Bishop, et al., 395/650; 364/229, 229.2,
230, 230.3, 280, 281, 281.3, 281.6, 281.7 [IMAGE AVAILABLE]

29.  5,109,515, Apr. 28, 1992, User and application program transparent
resource sharing multiple computer interface architecture with kernel process
level transfer of user requested services; George E. Laggis, et al., 395/725;
364/232.3, 235, 239, 241.9, 242.6, 242.7, 242.94, 247, 248.1, 253, 253.2,
254, 254.5, 265, 280, 280.2, 280.6, 280.9, 281.3, 284, DIG.1 [IMAGE
AVAILABLE]

41.  4,901,231, Feb. 13, 1990, Extended process for a multiprocessor system;
Thomas P. Bishop, et al., 395/325; 364/228.2, 230.3, 232.1, 245, 245.31, 247,
247.7, 256.3, 256.4, 280, 280.6, 280.9, 281, 281.3, 281.6, 281.8, 282, 284,
16:04:06 COPY AND CLEAR PAGE, PLEASE
```



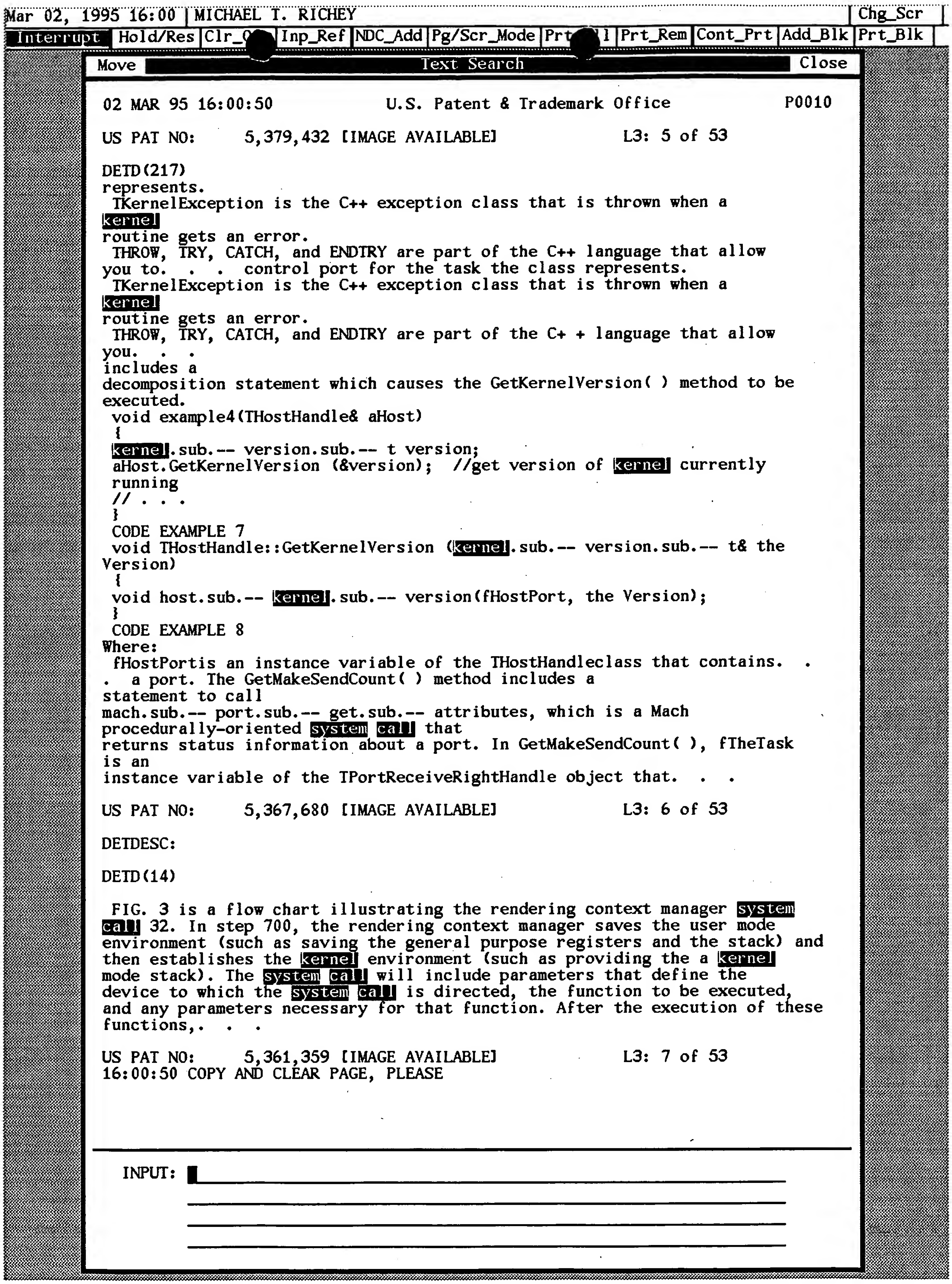
```
02 MAR 95 15:54:56          U.S. Patent & Trademark Office              P0003
                  23 SYSTEM PROCESS/AB
                     ((SYSTEM(W)PROCESS)/AB)
                 262 KERNEL/AB
L4                 2 ((SYSTEM CALL/AB OR SYSTEM PROCESS/AB)(P)(KERNEL/AB))

=> d 1-2

1.  4,761,737, Aug. 2, 1988, Method to automatically increase the segment
size of unix files in a page segmented virtual memory data processing system;
Keith E. Duvall, et al., 395/400; 364/256.3, 280.9, 283.1, DIG.1; 395/725
[IMAGE AVAILABLE]

2.  4,742,447, May 3, 1988, Method to control I/O accesses in a multi-tasking
virtual memory virtual machine type data processing system; Keith E. Duvall,
et al., 395/375; 364/DIG.1; 395/400, 425 [IMAGE AVAILABLE]

=> d hit 1-2

US PAT NO:      4,761,737 [IMAGE AVAILABLE]               L4: 1 of 2

ABSTRACT:
A memory management system method increases the size of a segment in blocks
of 64K virtual pages in response to the system detecting that the requested
page has been protected. The conventional UNIX type System Calls create and
open files in virtual memory. All pages are protected "read only" until a
SHMAT type System Call is made to operate on a page at a specific
address. At that point in the process, a protection exception is recognized
by the system and the UNIX kernel takes control to remove the protection
and update the appropriate data structures to reflect the new status of the
page and the addresses in real memory where the page may be found. Segments
containing mapped files are also extended by the method.

US PAT NO:      4,742,447 [IMAGE AVAILABLE]               L4: 2 of 2

ABSTRACT:
A method for accessing information in a page segmented virtual memory data
processing system in which virtual machines running UNIX type operating
systems are concurrently established, and in which a memory manager controls
the transfer of information between primary and secondary storage devices in
response to the occurrence of page faults.
The method establishes a plurality of data structures in a dynamic manner in
response to a Supervisor call to "map" a file. The mapping process assigns a
new segment of virtual memory to the mapped file and correlates, in one data
structure, the virtual address of each page of data in the new segment to a
disk file address where that page is actually stored.
A UNIX system call by an application program for a specific virtual page
is handled by the page fault hanger, and not the UNIX kernel, since the
application can supply the real address of the page on the disk file from the
data structure that was created by the mapped page range Supervisor call.
Simple load and store type of instructions are employed for the data
transfer, which avoids much of the overhead that normally accompanies
conventional UNIX read and write system calls to the storage subsystem.

=>
```



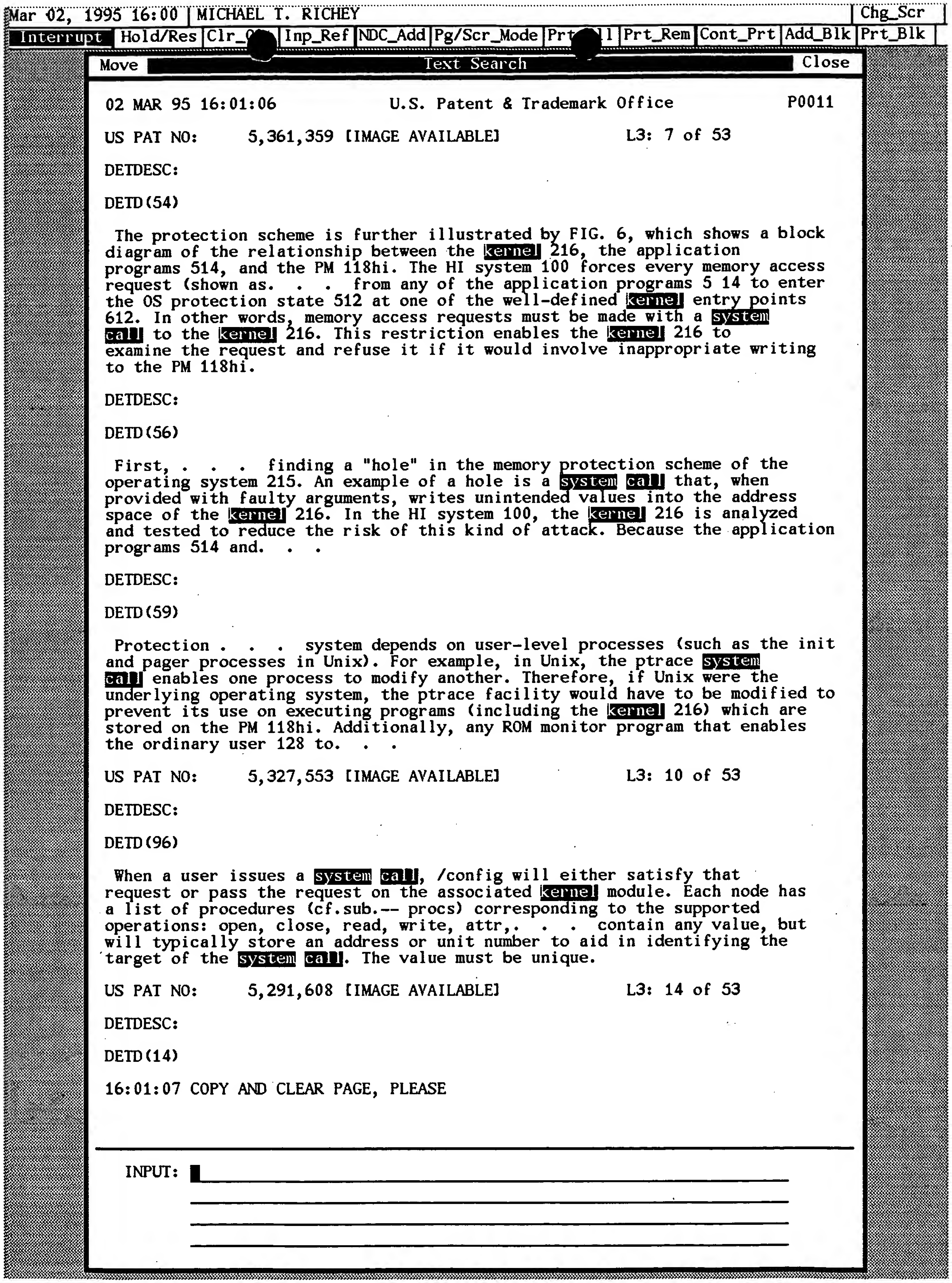
```
02 MAR 95 15:59:22              U.S. Patent & Trademark Office           P0008

46.  4,819,159, Apr. 4, 1989, Distributed multiprocess transaction processing
system and method; Dale L. Shipley, et al., 395/575; 364/230.6, 236.3, 238.4,
238.5, 239, 239.7, 242.4, 242.6, 242.92, 242.94, 242.95, 244, 244.8, 248.1,
265, 266.3, 267, 267.2, 267.4, 268, 268.3, 268.4, 268.5, 268.7, 268.9, 269.2,
269.3, 280, 281.3, 281.8, 282.1, 282.4, DIG.1; 371/9.1 [IMAGE AVAILABLE]

47.  4,769,771, Sep. 6, 1988, Multiprocessor system comprising a plurality of
data processors which are interconnected by a communication network; Wouter
J. H. M. Lippmann, et al., 395/200; 364/DIG.1; 395/650 [IMAGE AVAILABLE]

48.  4,761,737, Aug. 2, 1988, Method to automatically increase the segment
size of unix files in a page segmented virtual memory data processing system;
Keith E. Duvall, et al., 395/400; 364/256.3, 280.9, 283.1, DIG.1; 395/725
[IMAGE AVAILABLE]

49.  4,742,450, May 3, 1988, Method to share copy on write segment for mapped
files; Keith E. Duvall, et al., 395/700; 364/DIG.1 [IMAGE AVAILABLE]

50.  4,742,447, May 3, 1988, Method to control I/O accesses in a
multi-tasking virtual memory virtual machine type data processing system;
Keith E. Duvall, et al., 395/375; 364/DIG.1; 395/400, 425 [IMAGE AVAILABLE]

51.  4,649,479, Mar. 10, 1987, Device driver and adapter binding technique;
Hira Advani, et al., 395/700; 364/228.2, DIG.1 [IMAGE AVAILABLE]

52.  4,625,081, Nov. 25, 1986, Automated telephone voice service system;
Lawrence A. Lotito, et al., 379/88, 196, 211; 902/2, 39 [IMAGE AVAILABLE]

53.  4,519,032, May 21, 1985, Memory management arrangement for
microprocessor systems; Harry B. Mendell, 395/425; 364/231.4, 231.6, 232.8,
232.9, 236.2, 238.3, 238.4, 240.1, 241.2, 243, 244, 244.6, 246, 246.3, 246.6,
248.1, 252, 259, 259.2, 260.4, 260.8, 280, 280.6, 280.9, 286.4, 286.5, DIG.1;
395/725 [IMAGE AVAILABLE]

=> d kwic 2 5 6 7 10 14 18 27 29 41
       2 ANSWERS ARE AVAILABLE. SPECIFIED ANSWER NUMBER EXCEEDS ANSWER SET SIZE
YOU HAVE RECEIVED THIS ERROR MESSAGE  2 CONSECUTIVE TIMES
The answer numbers requested are not in the answer set.
IF YOU REQUIRE FURTHER HELP, PLEASE CONTACT YOUR LOCAL HELP DESK
ENTER ANSWER NUMBER OR RANGE (1):end

=> d 13 kwic 2 5 6 7 10 14 18 27 29 41

US PAT NO:      5,390,329 [IMAGE AVAILABLE]              L3: 2 of 53

SUMMARY:

BSUM(7)

 The kernel must save the register portion of the context of a process
whenever a process is switched out of execution, and.  .  .  shared. These
cycles impact system performance because register context switches are done
each time the system receives an interrupt, the kernel does a process
switch, or when a user makes a system call. To switch one process out and
another process in requires the current process's register context to be
saved, and the.  .  .  since user processes require all that context to do
work. However, a significant number of processes do only system or kernel
16:00:08 COPY AND CLEAR PAGE, PLEASE

INPUT: █
```



```
02 MAR 95 16:00:34          U.S. Patent & Trademark Office          P0009

US PAT NO:      5,390,329 [IMAGE AVAILABLE]              L3: 2 of 53

BSUM(7)
work. These kernel processes carry some amounts of unnecessary context, and
therefore take longer to create and to switch than they actually need to
take. There is a need in the prior art to develop a method by which kernel
functions are performed more efficiently and consume only the processor
resources absolutely required.

SUMMARY:

BSUM(12)

 The performance of kernel functions such as interrupt handling in prior
art systems has several basic problems. A major problem is the amount of
system overhead required to perform kernel functions. Context switches and
process start-up times consume processor cycles and affect the overall
performance of the processor. Efficiency is.  .  .  only one kind of process
entity can be created or processed by only one kind of creating entity, the
fork system call. As a result, the kernel processes contain more
context information than is actually required. The user-side context of
kernel processes (i.e., user block and various process table fields) is
ignored by the system and is not used. However, even.  .  .

SUMMARY:

BSUM(17)

 A .  .  .  invention introduces a new type of process called an iproc. An
iproc is a minimal-context process designed to efficiently execute kernel
tasks. Iprocs provide an efficient means of doing work created by an
interrupt or other system event into a processor.  .  .  prior to the event.
The inewproc procedure is called to create a minimal-context iproc in the
same way the fork system call is called to create a full-context process.
An advantage of the present invention is that because an iproc carries only.
.  .

US PAT NO:      5,379,432 [IMAGE AVAILABLE]              L3: 5 of 53

DETDESC:

DETD(58)

 Because the kernel guarantees both that port rights cannot be
counterfeited and that messages cannot be misdirected or falsified, port
rights provide a.  .  .  threads, memory objects, external memory managers,
permissions to do system-privileged operations, processor allocations, and so
on. In addition, since the kernel can send and receive messages itself (it
represents itself as a "special" task), the majority of the kernel services
are accessed via IPC messages instead of system-call traps. This has
allowed services to be migrated out of the kernel fairly easily where
appropriate.

DETDESC:

DETD(217)
 .  .  .
   control port for the thread the class
16:00:35 COPY AND CLEAR PAGE, PLEASE

 INPUT:
```



```
02 MAR 95 16:00:50          U.S. Patent & Trademark Office         P0010

US PAT NO:     5,379,432 [IMAGE AVAILABLE]             L3: 5 of 53

DETD(217)
represents.
 TKernelException is the C++ exception class that is thrown when a
kernel
routine gets an error.
 THROW, TRY, CATCH, and ENDTRY are part of the C++ language that allow
you to.  .  .  control port for the task the class represents.
 TKernelException is the C++ exception class that is thrown when a
kernel
routine gets an error.
 THROW, TRY, CATCH, and ENDTRY are part of the C+ + language that allow
you.  .  .
includes a
decomposition statement which causes the GetKernelVersion( ) method to be
executed.
 void example4(THostHandle& aHost)
 {
 kernel.sub.-- version.sub.-- t version;
 aHost.GetKernelVersion (&version);  //get version of kernel currently
 running
 // . . .
 }
 CODE EXAMPLE 7
 void THostHandle::GetKernelVersion (kernel.sub.-- version.sub.-- t& the
Version)
 {
 void host.sub.-- kernel.sub.-- version(fHostPort, the Version);
 }
 CODE EXAMPLE 8
Where:
 fHostPortis an instance variable of the THostHandleclass that contains.  .
.  a port. The GetMakeSendCount( ) method includes a
statement to call
mach.sub.-- port.sub.-- get.sub.-- attributes, which is a Mach
procedurally-oriented system call that
returns status information about a port. In GetMakeSendCount( ), fTheTask
is an
instance variable of the TPortReceiveRightHandle object that.  .  .

US PAT NO:     5,367,680 [IMAGE AVAILABLE]             L3: 6 of 53

DETDESC:

DETD(14)

 FIG. 3 is a flow chart illustrating the rendering context manager system
call 32. In step 700, the rendering context manager saves the user mode
environment (such as saving the general purpose registers and the stack) and
then establishes the kernel environment (such as providing the a kernel
mode stack). The system call will include parameters that define the
device to which the system call is directed, the function to be executed,
and any parameters necessary for that function. After the execution of these
functions,.  .  .

US PAT NO:     5,361,359 [IMAGE AVAILABLE]             L3: 7 of 53
16:00:50 COPY AND CLEAR PAGE, PLEASE

 INPUT:
```



Mar 02, 1995 16:00 | MICHAEL T. RICHEY | Chg_Scr

Interrupt | Hold/Res | Clr_ | Inp_Ref | NDC_Add | Pg/Scr_Mode | Prt ll | Prt_Rem | Cont_Prt | Add_Blk | Prt_Blk

Move | Text Search | Close

02 MAR 95 16:01:06 U.S. Patent & Trademark Office P0011

US PAT NO: 5,361,359 [IMAGE AVAILABLE] L3: 7 of 53

DETDESC:

DETD(54)

The protection scheme is further illustrated by FIG. 6, which shows a block diagram of the relationship between the kernel 216, the application programs 514, and the PM 118hi. The HI system 100 forces every memory access request (shown as. . . from any of the application programs 5 14 to enter the OS protection state 512 at one of the well-defined kernel entry points 612. In other words, memory access requests must be made with a system call to the kernel 216. This restriction enables the kernel 216 to examine the request and refuse it if it would involve inappropriate writing to the PM 118hi.

DETDESC:

DETD(56)

First, . . . finding a "hole" in the memory protection scheme of the operating system 215. An example of a hole is a system call that, when provided with faulty arguments, writes unintended values into the address space of the kernel 216. In the HI system 100, the kernel 216 is analyzed and tested to reduce the risk of this kind of attack. Because the application programs 514 and. . .

DETDESC:

DETD(59)

Protection . . . system depends on user-level processes (such as the init and pager processes in Unix). For example, in Unix, the ptrace system call enables one process to modify another. Therefore, if Unix were the underlying operating system, the ptrace facility would have to be modified to prevent its use on executing programs (including the kernel 216) which are stored on the PM 118hi. Additionally, any ROM monitor program that enables the ordinary user 128 to. . .

US PAT NO: 5,327,553 [IMAGE AVAILABLE] L3: 10 of 53

DETDESC:

DETD(96)

When a user issues a system call, /config will either satisfy that request or pass the request on the associated kernel module. Each node has a list of procedures (cf.sub.-- procs) corresponding to the supported operations: open, close, read, write, attr,. . . contain any value, but will typically store an address or unit number to aid in identifying the target of the system call. The value must be unique.

US PAT NO: 5,291,608 [IMAGE AVAILABLE] L3: 14 of 53

DETDESC:

DETD(14)

16:01:07 COPY AND CLEAR PAGE, PLEASE

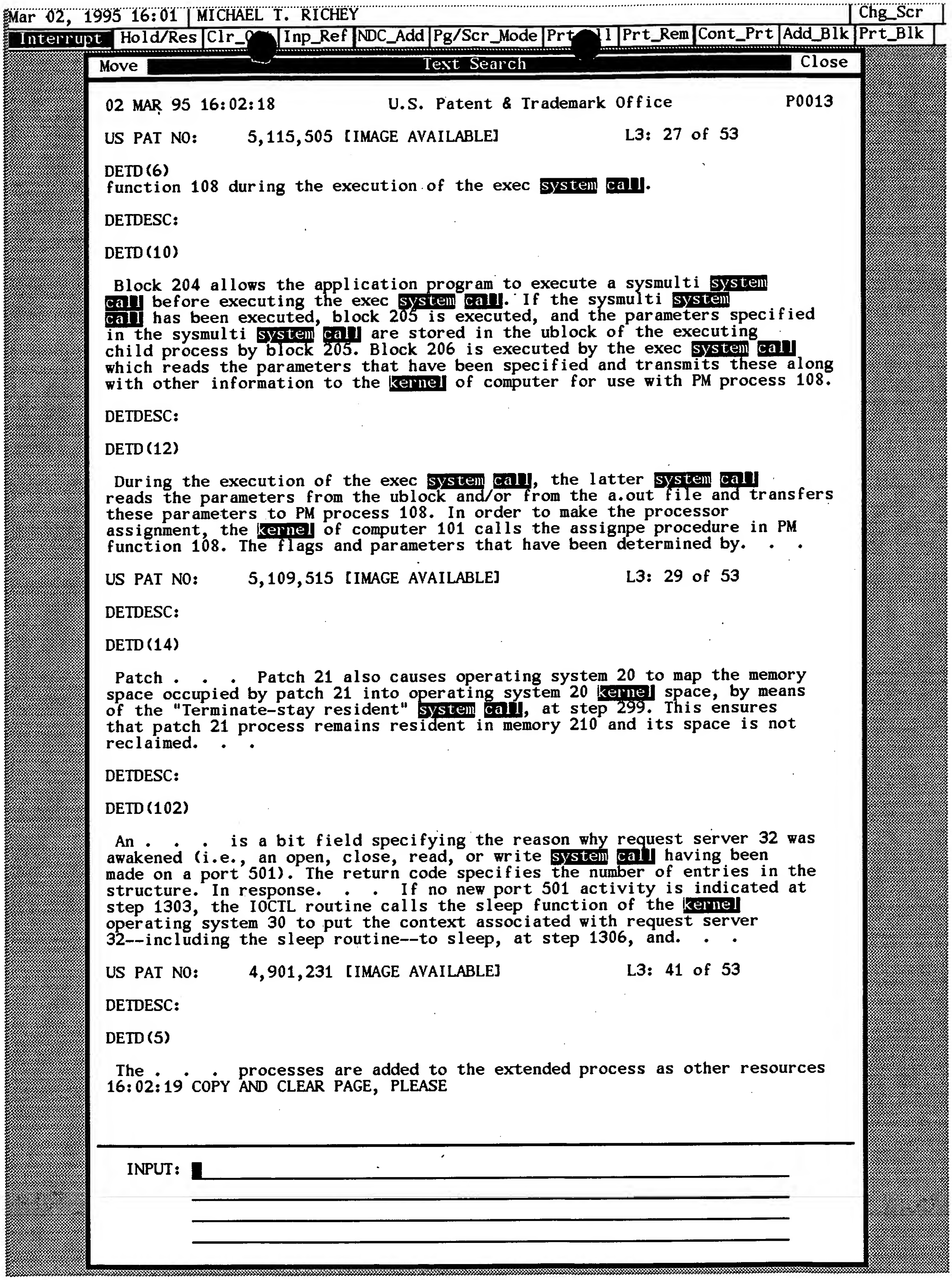
INPUT:

Mar 02, 1995 16:00 | MICHAEL T. RICHEY | Chg_Scr

Interrupt | Hold/Res | Clr_ | Inp_Ref | NDC_Add | Pg/Scr_Mode | Prt l | Prt_Rem | Cont_Prt | Add_Blk | Prt_Blk

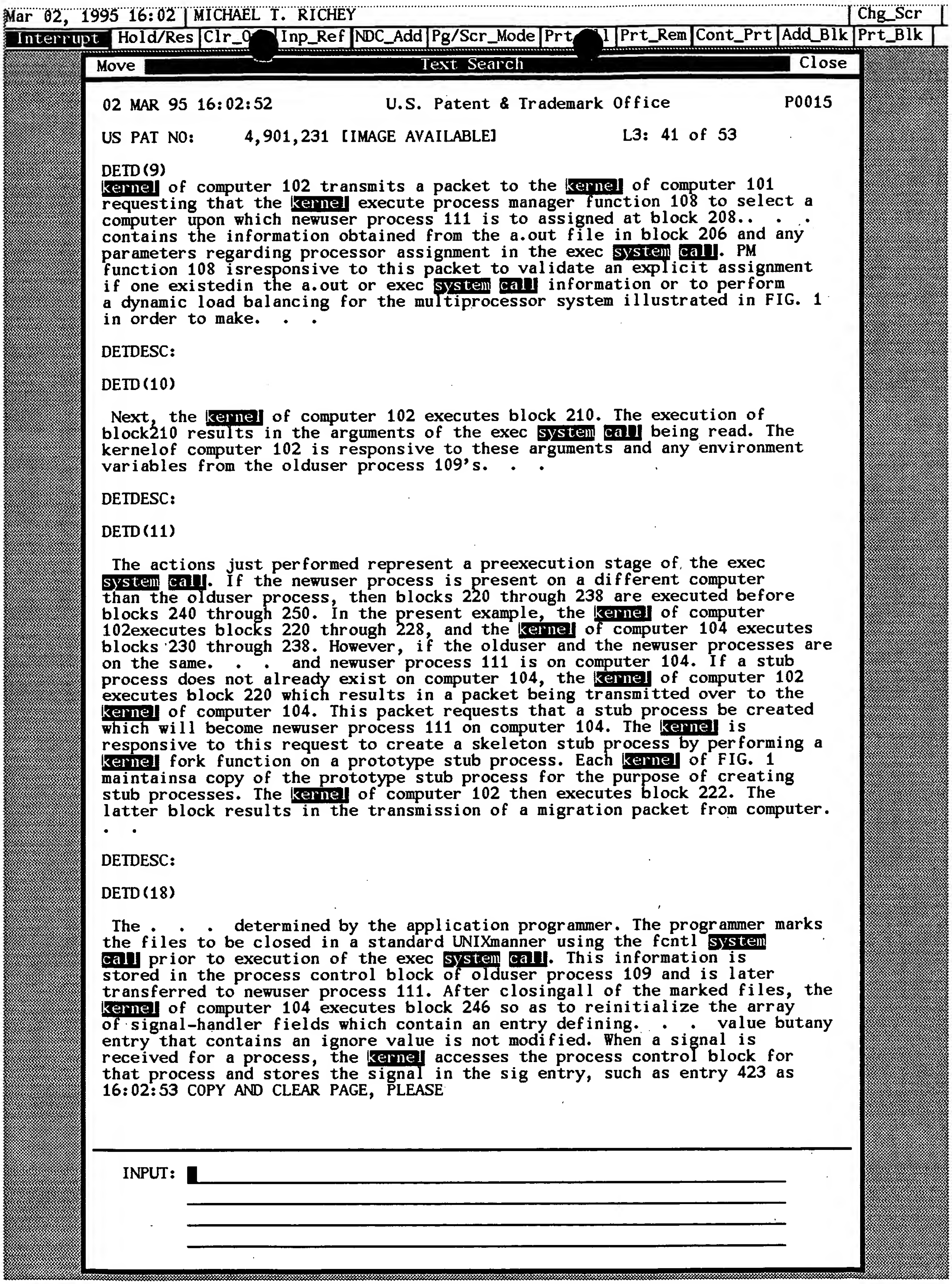
Move | Text Search | Close

```
02 MAR 95 16:01:26           U.S. Patent & Trademark Office           P0012

US PAT NO:      5,291,608 [IMAGE AVAILABLE]            L3: 14 of 53

DETD(14)
 FIG. 3 is a flow chart illustrating the rendering context manager system
call 32. In step 700, the rendering context manager saves the user mode
environment (such as saving the general purpose registers and the stack) and
then establishes the kernel environment (such as providing the a kernel
mode stack). The system call will include parameters that define the
device to which the system call is directed, the function to be executed,
and any parameters necessary for that function. After the execution of these
functions,.  .  .

US PAT NO:      5,202,971 [IMAGE AVAILABLE]            L3: 18 of 53

SUMMARY:

BSUM(28)

 In the standalone system, the kernel buffer 12 is identified by blocks 15
which are designated as device number and logical block number within the
device. When a read system call 16 is issued, it is issued with a file
descriptor of the file 5 and a byte range within the.  .  .

DETDESC:

DETD(93)

 Files,  .  .  .  advisory locking mode. The locking mode of a file is
controlled by changing the file's permission codes with the chmod system
call. Locks on a file in enforced locking mode are called enforced locks;
locks on a file in advisory mode are.  .  .  that they are accessing. The
advantage of advisory locks is that they do not have to be interrogated by
the kernel during ordinary reading or writing operations. Enforced locks
will probably be used less often. An enforced lock, like an advisory.  .  .

US PAT NO:      5,115,505 [IMAGE AVAILABLE]            L3: 27 of 53

DETDESC:

DETD(2)

 FIG.  .  .  .  assignment, allocation of resources and dynamic load balancing
is performed by process manager (PM) function 108 being executed by the
kernel of computer 101. The latter computer is designated as the host
computer of the system, FIG. 1. The execution of an exec system call by
any computer of FIG. 1 results in PM function 108 being executed by the
kernel of computer 101. The latter function's use of flags, variables, and
parameters that are specified by the system administrator, application.  .  .

DETDESC:

DETD(6)

 The  .  .  .  dynamic loading function at the time the source code of the
requesting program is written. Upon execution of the sysmulti system
call, the parameters defining the application programmer's adjustments are
initially stored in the ublock of the process in a standard manner. These
parameters are then passed to the kernel of computer 101 for use with PM
16:01:27 COPY AND CLEAR PAGE, PLEASE

INPUT:
```



```
02 MAR 95 16:02:18             U.S. Patent & Trademark Office              P0013

US PAT NO:      5,115,505 [IMAGE AVAILABLE]               L3: 27 of 53

DETD(6)
function 108 during the execution of the exec system call.

DETDESC:

DETD(10)

 Block 204 allows the application program to execute a sysmulti system
call before executing the exec system call. If the sysmulti system
call has been executed, block 205 is executed, and the parameters specified
in the sysmulti system call are stored in the ublock of the executing
child process by block 205. Block 206 is executed by the exec system call
which reads the parameters that have been specified and transmits these along
with other information to the kernel of computer for use with PM process 108.

DETDESC:

DETD(12)

 During the execution of the exec system call, the latter system call
reads the parameters from the ublock and/or from the a.out file and transfers
these parameters to PM process 108. In order to make the processor
assignment, the kernel of computer 101 calls the assignpe procedure in PM
function 108. The flags and parameters that have been determined by.  .  .

US PAT NO:      5,109,515 [IMAGE AVAILABLE]               L3: 29 of 53

DETDESC:

DETD(14)

 Patch .  .  .  Patch 21 also causes operating system 20 to map the memory
space occupied by patch 21 into operating system 20 kernel space, by means
of the "Terminate-stay resident" system call, at step 299. This ensures
that patch 21 process remains resident in memory 210 and its space is not
reclaimed.  .  .

DETDESC:

DETD(102)

 An .  .  .  is a bit field specifying the reason why request server 32 was
awakened (i.e., an open, close, read, or write system call having been
made on a port 501). The return code specifies the number of entries in the
structure. In response.  .  .  If no new port 501 activity is indicated at
step 1303, the IOCTL routine calls the sleep function of the kernel
operating system 30 to put the context associated with request server
32--including the sleep routine--to sleep, at step 1306, and.  .  .

US PAT NO:      4,901,231 [IMAGE AVAILABLE]               L3: 41 of 53

DETDESC:

DETD(5)

 The .  .  .  processes are added to the extended process as other resources
16:02:19 COPY AND CLEAR PAGE, PLEASE


   INPUT: █
```

```
Mar 02, 1995 16:02 | MICHAEL T. RICHEY                                          | Chg_Scr
Interrupt | Hold/Res | Clr_Q | Inp_Ref | NDC_Add | Pg/Scr_Mode | Prt_All | Prt_Rem | Cont_Prt | Add_Blk | Prt_Blk
Move                              Text Search                                  Close

02 MAR 95 16:02:31          U.S. Patent & Trademark Office          P0014

US PAT NO:      4,901,231 [IMAGE AVAILABLE]          L3: 41 of 53

DETD(5)
are required in different processors of the multiprocessor system. The
kernel of the processor executing the user process of the extended process
automatically detects the need to create a stub process when a system
call is made by the user process requiring resources on a new processor.
The user process's kernel then communicates with the kernelof the new
processor to establish a stub process on the new processor. The establishment
of.  .  .

DETDESC:

DETD(7)

 Upon olduser process 109 executing the exec system call, the kernel of
computer 102 transmits a packet to computer 103 to obtain the header portion
of the a.out file via a.out.  .  .  108 being executed by computer 101 which
is designated as the host computer of the system of FIG. 1. The kernel of
computer 102 then transmits a packet to process manager function 108 of
computer 101 requesting allocation of resources for.  .  .  the operations of
process manager function 108 is illustrated in the copending application of
Bishop et al., Ser. No.941,701. The kernel of computer 102 then transmits
process control information to the kernel of computer 104 so that the
latter kernel can setup newuser process 111 and stub processes in computers
102, 105, and 106 for the future execution of the.  .  .

DETDESC:

DETD(8)

 Once this initialization has been performed, the kernel of computer 102
passes the execution of the exec system call to the kernel of computer
104. The latter kernel obtains the a.out file from computer 103. The
kernel of computer 104 also transmits messages to the kernels of the
othercomputers informing them that the user process which was.  .  .  kernels
of the other computers will now direct any signals for olduser process 109 to
newuser process 111. Further, the kernel of computer 104 transmits a
message to the kernel of computer 102 to recover all signals transmitted to
olduser process 109 that arrived at computer 102 before the other.  .  .

DETDESC:

DETD(9)

 FIG. 2 illustrates in greater detail the execution of the exec system
call and creation of the extended process for the present example. Upon
execution of the exec system call by olduser process 109, decision block
202 is performed. The exec system call may specify parameters for
influencing the processor assignment. Decision block 202 determines whether
or not the file containing the a.out.  .  .  extended process, a packet is
sent to create a stub process on computer 103. In response to the packet, the
kernel of computer 103 creates a.out process 110 that allows access to
thea.out file. The a.out process 110 then becomes part.  .  .  information is
read from the a.out file and is stored in the process control block of a.out
process 110. The kernel ofcomputer 103 then transmits a subset of the
header to computer 102's kernelwhich stores the subset in the process
control.  .  .  file and may specify parameters for influencing the processor
assignment decision. After obtaining the information from the a.out file, the
16:02:31 COPY AND CLEAR PAGE, PLEASE

INPUT:
```



```
02 MAR 95 16:02:52              U.S. Patent & Trademark Office              P0015

US PAT NO:      4,901,231 [IMAGE AVAILABLE]              L3: 41 of 53

DETD(9)
kernel of computer 102 transmits a packet to the kernel of computer 101
requesting that the kernel execute process manager function 108 to select a
computer upon which newuser process 111 is to assigned at block 208..  .  .
contains the information obtained from the a.out file in block 206 and any
parameters regarding processor assignment in the exec system call. PM
function 108 isresponsive to this packet to validate an explicit assignment
if one existedin the a.out or exec system call information or to perform
a dynamic load balancing for the multiprocessor system illustrated in FIG. 1
in order to make.  .  .

DETDESC:

DETD(10)

 Next, the kernel of computer 102 executes block 210. The execution of
block210 results in the arguments of the exec system call being read. The
kernelof computer 102 is responsive to these arguments and any environment
variables from the olduser process 109's.  .  .

DETDESC:

DETD(11)

 The actions just performed represent a preexecution stage of the exec
system call. If the newuser process is present on a different computer
than the olduser process, then blocks 220 through 238 are executed before
blocks 240 through 250. In the present example, the kernel of computer
102executes blocks 220 through 228, and the kernel of computer 104 executes
blocks 230 through 238. However, if the olduser and the newuser processes are
on the same.  .  .  and newuser process 111 is on computer 104. If a stub
process does not already exist on computer 104, the kernel of computer 102
executes block 220 which results in a packet being transmitted over to the
kernel of computer 104. This packet requests that a stub process be created
which will become newuser process 111 on computer 104. The kernel is
responsive to this request to create a skeleton stub process by performing a
kernel fork function on a prototype stub process. Each kernel of FIG. 1
maintainsa copy of the prototype stub process for the purpose of creating
stub processes. The kernel of computer 102 then executes block 222. The
latter block results in the transmission of a migration packet from computer.
.  .

DETDESC:

DETD(18)

 The .  .  .  determined by the application programmer. The programmer marks
the files to be closed in a standard UNIXmanner using the fcntl system
call prior to execution of the exec system call. This information is
stored in the process control block of olduser process 109 and is later
transferred to newuser process 111. After closingall of the marked files, the
kernel of computer 104 executes block 246 so as to reinitialize the array
of signal-handler fields which contain an entry defining.  .  .  value butany
entry that contains an ignore value is not modified. When a signal is
received for a process, the kernel accesses the process control block for
that process and stores the signal in the sig entry, such as entry 423 as
16:02:53 COPY AND CLEAR PAGE, PLEASE



INPUT:
```

Mar 02, 1995 16:02 | MICHAEL T. RICHEY | Chg_Scr

Interrupt | Hold/Res | Clr_O[illegible] | Inp_Ref | NDC_Add | Pg/Scr_Mode | Prt[illegible]l | Prt_Rem | Cont_Prt | Add_Blk | Prt_Blk

Move | Text Search | Close

02 MAR 95 16:03:10 U.S. Patent & Trademark Office P0016

US PAT NO: 4,901,231 [IMAGE AVAILABLE] L3: 41 of 53

DETD(18)
illustrated in FIG. 4. When a signal is handled by the kernel, the signal number is used as an index to access the signal array. If the default value is accessed, the. . . the accessed entrycontains a pointer, then the function identified by the pointer is executed. When the application runs, the signal system call will be used to configure the array to the requirements of that program. Further information on the handling of signals can be found in the aforementioned book of Bach. Next, the kernel of computer 104 executes block 248 which reinitializes any memory management information required for newuser process 111's new address space. . .

DETDESC:

DETD(19)

FIG. 3 illustrates in greater detail a portion of the extended process resulting from the execution of the exec system call. New process 111 is the user process of the extended process and processes 112 and 110 are stub processes of. . . maintains a proc pointer table such as 301 through 303 of FIG. 3. The pid number is utilized by the kernel to point into a proc pointer table such as tables 301 through 303 to obtain the pointer such as 304. . .

DETDESC:

DETD(27)

In . . . a.out file, and entry 432 points to the new a.out file which is executed as a result of the exec system call. If both files are local to the processor executing the user process of the extended process,then these entries are pointers which point into the inode table maintainedby the kernel of the local processor in a standard UNIX system manner to identify the local files. The system file table is not used since the a.out files are not used by the process directly but rather by the kernel.However, if the file is remote, e.g. associated with another processor, theentry contains an identification of an entry in the user. . .

DETDESC:

DETD(28)

In . . . stub process, the information concerning the path name is received from the user process and is then stored by the kernel at a convenient location. Atthis point and time, the kernel sets the dirp entry to point to the path name. An example of when the path name is transmitted is during an open system call. Since the open system call is executed on the processor executing the user process of the extended process, the path name information is not available. . .

=>

INPUT:

```
ProQuest (R)                       CD-ROM Retrieval                   Version 4.31
IEEE/IEE Publications Ondisc Jan 1988 - Nov 1994        F1=Help    F2=Commands

Öááááááááááááááááááááááááááááááá Search Entry ááááááááááááááááááááááááááááááá
º                                                                            Ü
º                                                                            _
º                                                                            _
º                                                                            _
âáááááááááááááááááááááááááááááááááááááááááááááááááááááááááááááááááááááááááááá


Öááááááááááááááááááááááááááááááááá Results áááááááááááááááááááááááááááááááááá¢
º Num    Search                                                          Hits º
º                                                                             º
º #1     system call                                                       21 º
º #2     kernel                                                           764 º
º #3     kernel w/30 #1                                                     2 º
º #4     kernel and #1                                                      2 º
º                                                                             º
º                                                                             º
º                                                                             º
º                                                                             º
âáá Press TAB to access previous results áááááááááááááááááááááááááááááááááááì

Type your search and press ENTER.      Press F5 for Periodical Directory.
```

```
                    Ei COMPENDEX*PLUS  January - December 1992
1   273 Records with Word/Phrase Index of KERNEL
2   92 Records remaining, Limiting to those with Word/Phrase Index of SYSTEM





              èëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëë£
              ¤                Select Main Activity                ¤
              äëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëë¿
              ¤ Display, Print or Transfer the Selected Record(s)  ¤
              ¤ Modify the current search with additional criteria ¤
              ¤ Begin a New Search (clears existing search)        ¤
              ¤ Save Search Strategy for Later Use                 ¤
              ¤ Quit Easy Menu Mode                                ¤
              àëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëëë¥

F1-Help   --Move  áì-Select   F2-Usage  F6-Databases  F8-PrtHist  F3-Restart
```